Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

INFORMATYKA STOSOWANA

SZTUCZNE SIECI NEURONOWE

SIEĆ ROZPOZNAJĄCA GESTY DŁONI

MAREK CABAJ, KAMIL KRÓL

Kraków 2012

Spis treści

1. Wstęp

1.1. Wprowadzenie i cel projektu

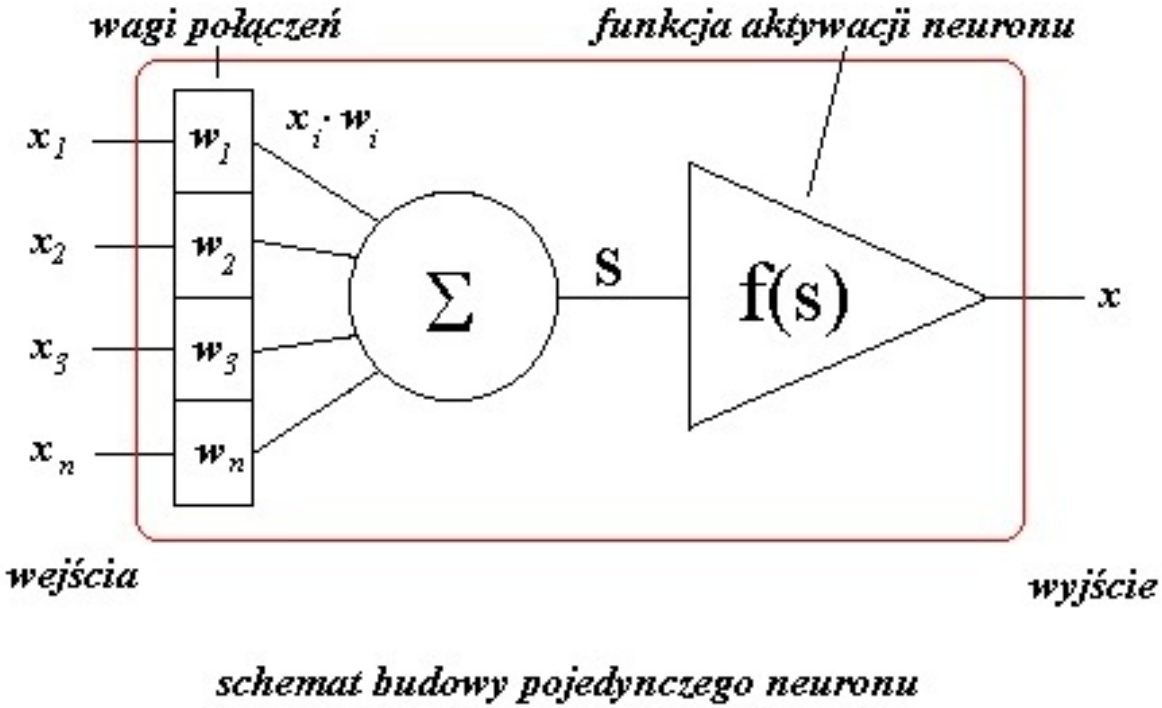
Celem projektu było stworzenie aplikacji umożliwiającej rozpoznawanie gestów dłoni w czasie rzeczywistym. Aplikacja w sposób ciągły analizowałaby obraz z kamery podłączonej do komputera i wraz z naciśnięciem odpowiedniego przycisku wyzwalającego przetwarzałaby otrzymany obraz. Następnie odpowiednio przetworzony obraz byłby przekazywany na wejście zaprojektowanej uprzednio sieci neuronowej, której zadaniem jest sprawdzenie jaki gest został pokazany (Otwarta dłoń, Zamknięta dłoń, Gest "V"). Aplikacja informowałaby użytkownika o rozpoznanym geście odpowiednim komunikatem.

1.2. Teoretyczne podstawy działania sztucznych sieci neuronowych

Sztuczne sieci neuronowe znajdują zastosowanie w sytuacjach, gdy zmuszeni jesteśmy realizować operacje typu „rozpoznawanie” lub „kojarzenie”. Podczas gdy dla klasycznych algorytmów nawet nieznaczna zmiana parametrów wejściowych rozpoznawanego obiektu stanowi dużą trudność, sieci neuronowe posiadają zdolność generalizacji czyli uogólniania wiedzy dla nowych danych nieznanych wcześniej, czyli nie prezentowanych w trakcie nauki. Określa się to także jako zdolność sieci neuronowych do aproksymacji wartości funkcji wielu zmiennych w przeciwieństwie do interpolacji możliwej do otrzymania przy przetwarzaniu algorytmicznym.

Najczęstszymi obszarami wykorzystania sztucznych sieci neuronowych są:

- Rozpoznawanie wzorców (znaków, liter, kształtów, sygnałów mowy, sygnałów sonarowych)
- Klasyfikowanie obiektów
- Prognozowanie zmian cen rynkowych (giełdy, waluty)
- Prognozowanie zapotrzebowania na energię elektryczną
- Diagnostyka medyczna
- Prognozowanie sprzedaży



Rysunek 1.1: Schemat budowy pojedynczego neuronu

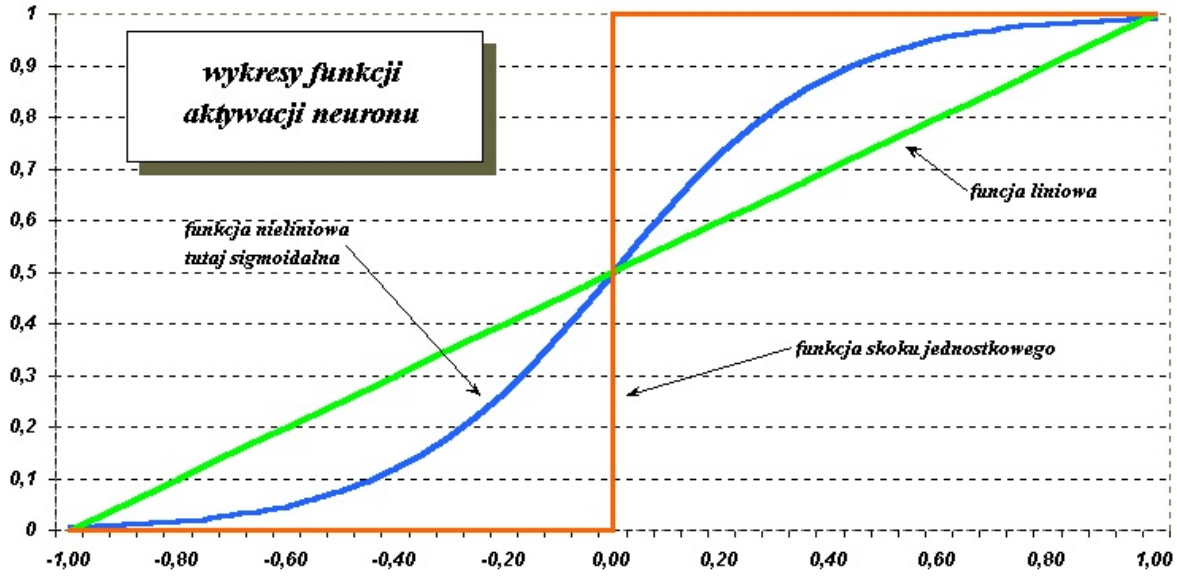
- Aproksymowanie wartości funkcji

Podstawowym elementem sieci neuronowej jest neuron. Jego schemat został przedstawiony na rysunku 1.1.

Do wejść wprowadzane są sygnały z warstwy poprzedniej, a każdy z nich mnożony jest przez wartość nazywaną wagą. To od niej zależy udział sygnału wejściowego w tworzeniu sygnału wyjściowego. Może być ona pobudzająca (dodatnia) lub opóźniająca (ujemna). Funkcja aktywacji neuronu jako argument przyjmuje zsumowane iloczyny sygnałów i wag. Może ona przyjąć jedną z trzech postaci, tak jak zostało to przedstawione na rysunku 1.2.

- Funkcja liniowa
- Funkcja nieliniowa
- Skok jednostkowy

Istnieją różne metody uczenia sieci neuronowych. W przygotowanej przez nas sieci korzystamy z metody wstecznej propagacji błędów. Jest to uczenie z nadzorem lub inaczej - z nauczycielem. Pierwszą czynnością w procesie uczenia jest przygotowanie dwóch ciągów danych: uczącego i weryfikującego. Ciąg uczący jest to zbiór takich danych, które w miarę dokładnie charakteryzują dany problem. Jednorazowa porcja danych nazywana jest wektorem



Rysunek 1.2: Funkcja aktywacji neuronu

uczącym. W jego skład wchodzi wektor wejściowy czyli te dane wejściowe, które podawane są na wejścia sieci i wektor wyjściowy czyli takie dane oczekiwane, jakie sieć powinna wygenerować na swoich wyjściach. Po przetworzeniu wektora wejściowego, nauczyciel porównuje wartości otrzymane z wartościami oczekiwanymi i informuje sieć czy odpowiedź jest poprawna, a jeżeli nie, to jaki powstał błąd odpowiedzi. Błąd ten jest następnie propagowany do sieci ale w odwrotnej niż wektor wejściowy kolejności (od warstwy wyjściowej do wejściowej) i na jego podstawie następuje taka korekcja wag w każdym neuronie, aby ponowne przetworzenie tego samego wektora wejściowego spowodowało zmniejszenie błędu odpowiedzi. Procedurę taką powtarza się do momentu wygenerowania przez sieć błędu mniejszego niż założony. Wtedy na wejście sieci podaje się kolejny wektor wejściowy i powtarza te czynności. Po przetworzeniu całego ciągu uczącego (proces ten nazywany jest epoką) oblicza się błąd dla epoki i cały cykl powtarzany jest do momentu, aż błąd ten spadnie poniżej dopuszczalnego. Jak to już było zasygnalizowane wcześniej, SSN wykazują tolerancję na nieciągłości, przypadkowe zaburzenia lub wręcz niewielkie braki w zbiorze uczącym. Jest to wynikiem właśnie zdolności do uogólniania wiedzy.

2. Realizacja aplikacji

Realizację projektu można podzielić na następujące etapy :

- Wybór technologii do wykonania aplikacji
- Podłączenie kamery i przechwytywanie z niej obrazu w czasie rzeczywistym
- Przetwarzanie przechwyconego obrazu
- Przygotowanie prostego GUI
- Implementacja odpowiedniej sieci neuronowej
- Przygotowanie zestawu uczącego i nauka sieci neuronowej
- Użycie utworzonej sieci neuronowej w celu rozpoznania gestu

2.1. Wybór technologii

Na potrzeby realizacji niniejszej aplikacji wybrany został pakiet Matlab, głównie ze względu na gotowe, bardzo funkcjonalne i proste w użyciu narzędzia do wstępnego przetwarzania obrazu oraz ze względu na intuicyjny i łatwo rekonfigurowalny pakiet Neural Networks użyty do tworzenia sieci neuronowej. Dodatkowym atutem była dobra znajomość pakietu Matlab przez twórców aplikacji.

2.2. Podłączenie kamery i przechwytywanie obrazu

Do przechwytywania obrazu został użyty standardowo dodany do Matlaba pakiet 'Image Acquisition Toolbox'. Aplikacja po uruchomieniu sprawdza jakie formaty video oraz jakie rozdzielczości są wspierane przez kamerę użytkownika a następnie tworzy obiekt 'videoinput', który w dalszej części aplikacji używany jest do pobierania obrazu w czasie rzeczywistym a także do przechwytywania pojedynczych klatek po naciśnięciu odpowiedniego przycisku. Pomimo chęci zaimplementowania pełnej automatyzacji w rozpoznawaniu typu kamery użytkownika, aplikacja nie jest kompatybilna ze wszystkimi urządzeniami.

2.3. Przetwarzanie przechwyconego obrazu

Klasyfikacja otrzymanego wzorca wymaga, aby pobrany obraz został wcześniej odpowiednio przetworzony. Proces ten składa się z kilku etapów:

- konwersja obrazu do skali szarości
- obliczenie odpowiedniego progu binaryzacji przy pomocy funkcji 'graythresh'
- binaryzacja z wyliczonym progiem
- czyszczenie krawędzi
- usunięcie wszystkich obiektów o powierzchni mniejszej niż 0,1 * pow. obrazu
- przeprowadzenie operacji zamknięcia
- przeskalowanie obrazu do rozdzielczości 20x15

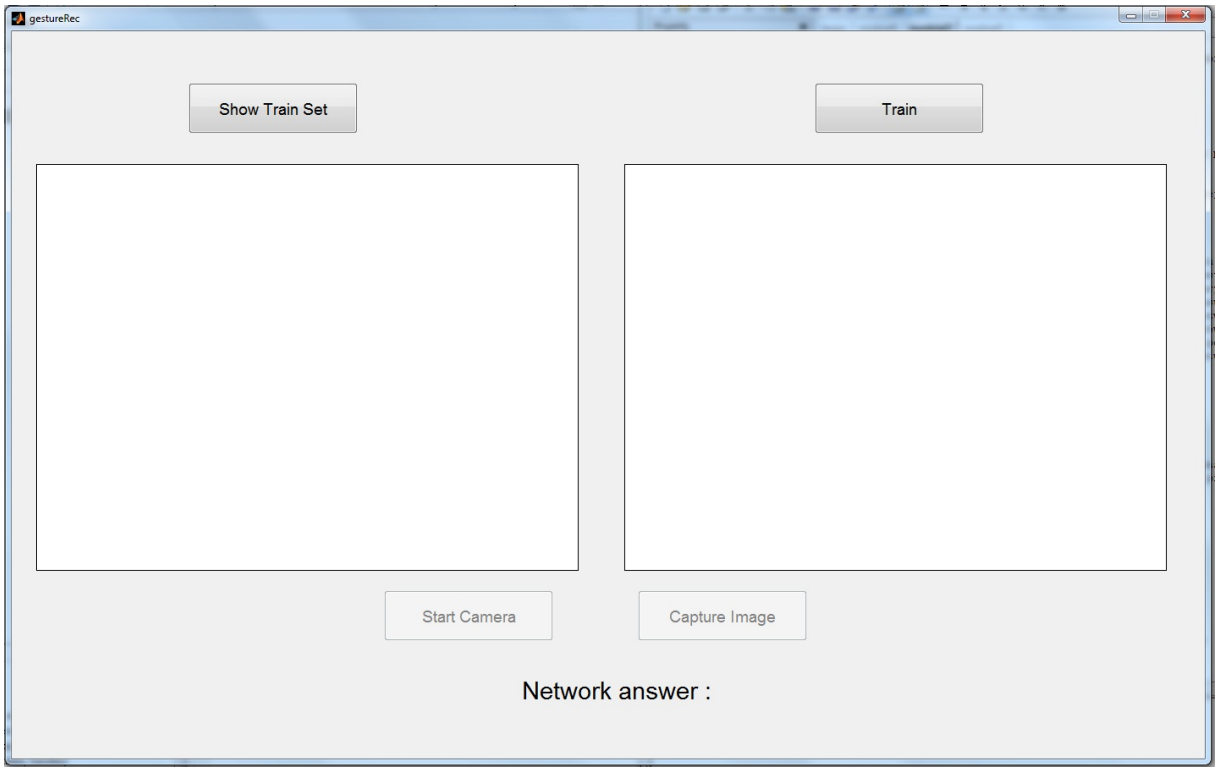
Do przetwarzania obrazu użyto gotowe funkcje z pakietu 'Image Processing Toolbox'. W projekcie przyjęto założenie, iż dłoń wykonująca gest będzie znajdowała się na jednorodnym, ciemnym tle umożliwiającym poprawną binaryzację obrazu. Przetworzenie obrazu pozwoliło na odróżnienie tła od obiektu oraz usunięcie prawie wszystkich zakłóceń. Ostateczna postać obrazu jest binarna co pozwala łatwo wykorzystać go jako wektor wejściowy dla sieci neuronowej. Jest on również pomniejszony co znacząco zwiększa wydajność procesu uczenia się, nie powodując odczuwalnego spadku jakości rozpoznawania wzorców. W celu poprawnego przetworzenia obrazu dłoń musi się znajdować w centrum obrazu.

2.4. Przygotowanie prostego GUI

W celu przechwytywania oraz przetwarzania obrazu w czasie rzeczywistym stworzono prosty graficzny interfejs użytkownika, który został przedstawiony na rysunku 2.1. Użytkownik ma do dyspozycji 4 przyciski oraz dwa pola z obrazem. Pole po lewej stronie pokazuje obraz z kamery w czasie rzeczywistym, natomiast pole po prawej przedstawia przechwycone oraz przetworzony obraz, który zostaje umieszczony na wejściu sieci neuronowej. Przyciski zostaną dokładniej opisane w kolejnych rozdziałach.

2.5. Przygotowanie zestawu uczącego i nauka sieci neuronowej

W niniejszej aplikacji użyto sieci neuronowej typu 'feedforward', która posiada 300 neuronów wejściowych (20 * 15 pikseli), 1 warstwę ukrytą, która liczy 40 neuronów oraz 3 neurony



Rysunek 2.1: Graficzny interfejs użytkownika

wyjściowe, które reprezentują otrzymany wynik (otwarta dłoń, pięść, znak 'V'). Liczbę neuronów w warstwie pośredniej określono na podstawie testów. Jak się okazało, zbyt duża ilość neuronów powoduje w sposób naturalny znaczące wydłużenie czasu nauki, a co ciekawsze, prowadzi do gorszego rozpoznawania nieznanych wzorców w przyszłości. Nieco mniejsza ilość neuronów pozwala na lepszą generalizację, a co za tym idzie na bardziej trafne określanie wyjścia.

Do stworzenia zestawu testowego użyto zmodyfikowanej aplikacji która zamiast kierować przetworzony obraz na wejście sieci neuronowej zapisywała go w pliku. Następnie ręcznie określane i dopisywane były poprawne gesty, które sieć powinna rozpoznać.

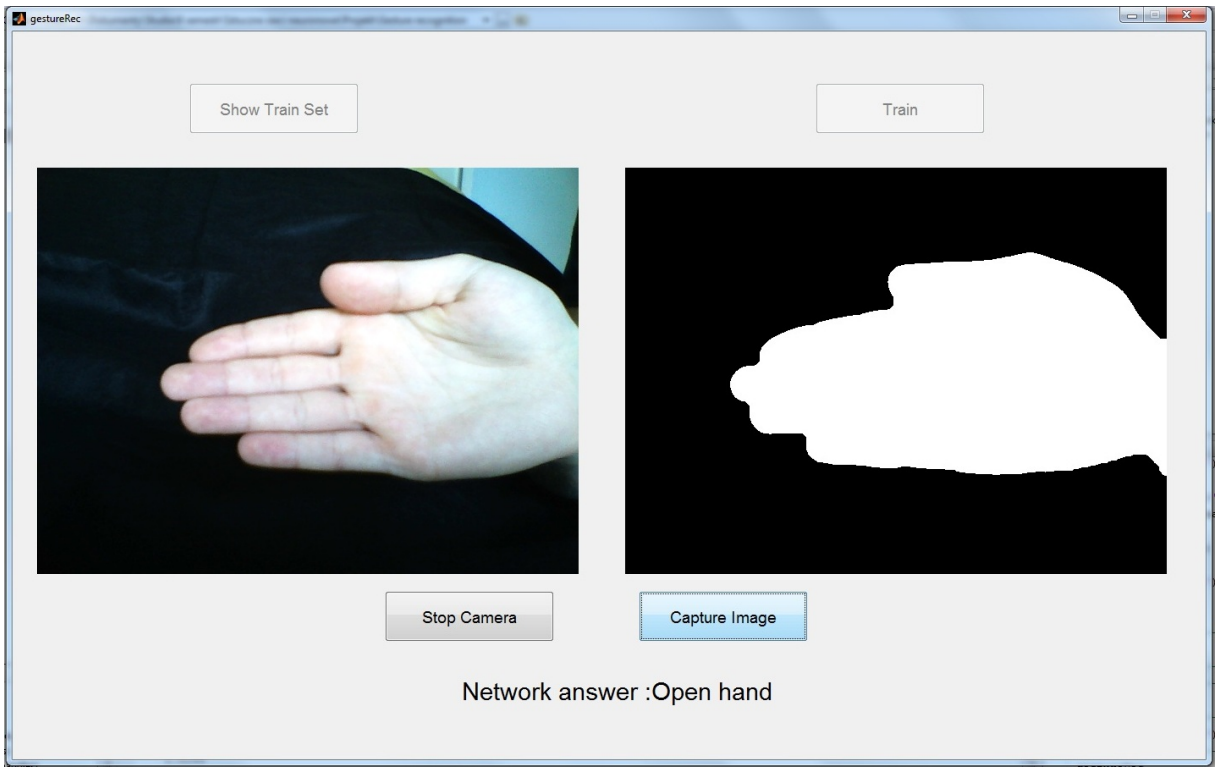
Przycisk 'Show train set' otwiera w nowym oknie przegląd całego zestawu, który zostanie użyty do nauki sieci neuronowej. W przypadku niniejszej aplikacji użyto zestawu treningowego złożonego z 15 obrazków, wraz z określeniem jaki to gest (po 5 obrazków na każdy z trzech gestów).

Przycisk 'Train' służy do uruchomienia narzędzia do nauki sieci neuronowej. Kiedy potrzebna ilość iteracji zostanie wykonana, a uzyskany błąd nauki będzie odpowiednio niski możemy nacisnąć przycisk 'Ok'. Aby nie było konieczności trenowania sieci przy każdym uruchomieniu, została ona zapisana w pliku 'net.mat'. Aby korzystać z gotowej sieci, a nie tworzyć i trenować nową należy zamienić 235 linijkę w pliku 'gestureRec.m' z 'network = neuralNetwork()' na 'network = getNetwork()'.

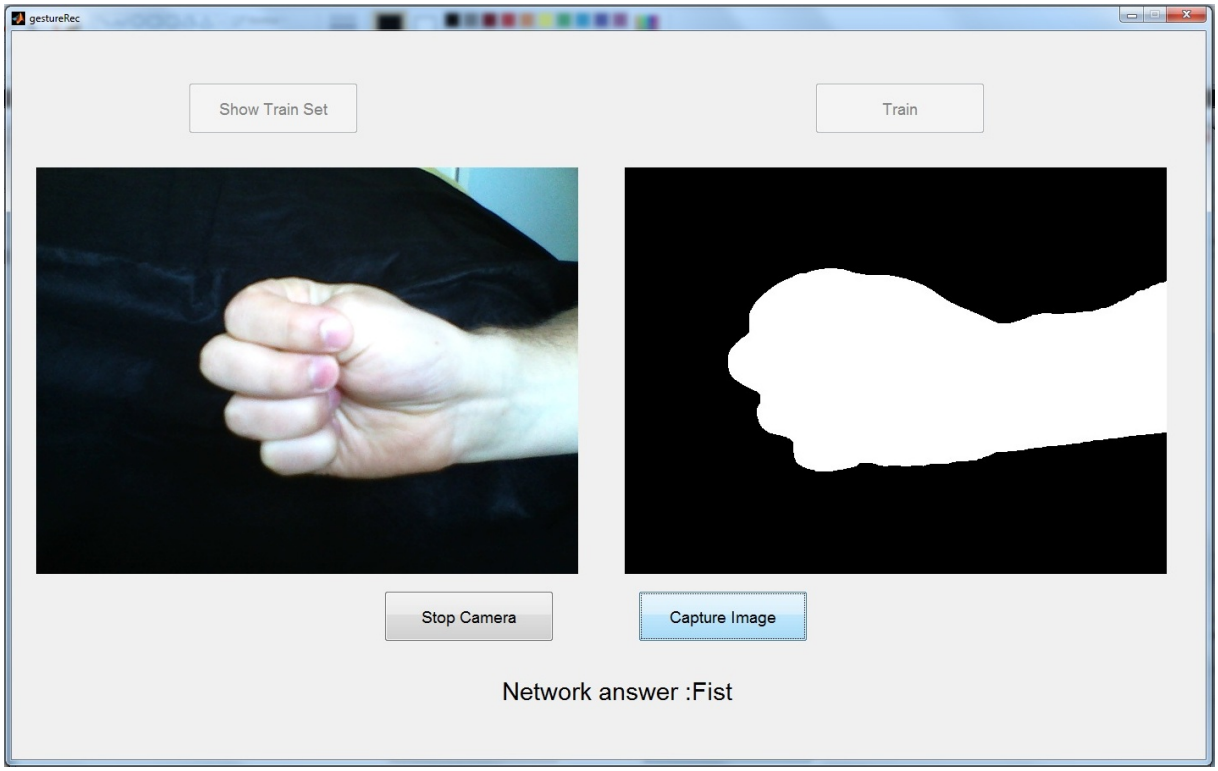
Po wykonaniu poprawnej nauki sieci zostaje odblokowany przycisk ’Start Camera’.

2.6. Użycie utworzonej sieci neuronowej w celu rozpoznania gestu

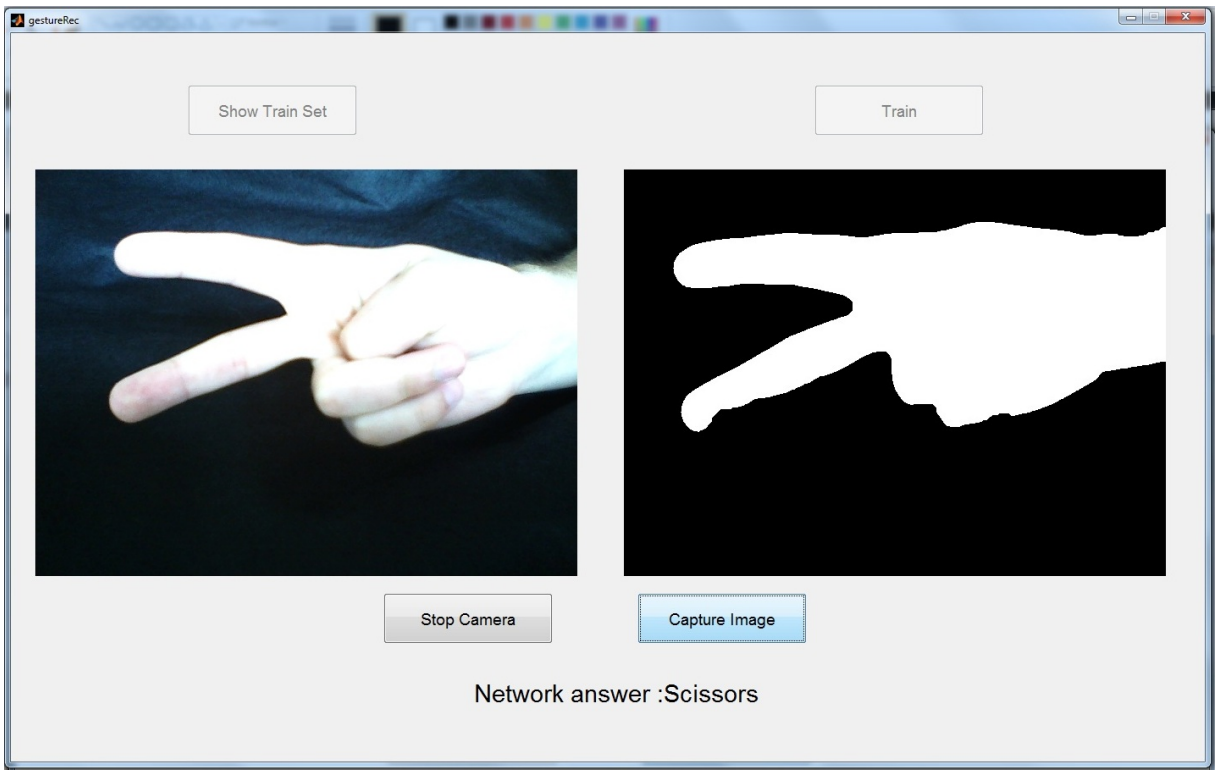
Aby rozpoznać przedstawiony gest należy najpierw uruchomić odczyt obrazu z kamery w czasie rzeczywistym. W tym celu naciskamy przycisk ’Start camera’ (pracę kamery możemy zatrzymać w dowolnym momencie przyciskiem ’Stop camera’). Kiedy kamera jest już aktywna i widzimy w lewym oknie przechwytywany obraz naciskamy przycisk ’Capture image’ w celu zrobienia zdjęcia. Tak otrzymany obrazek jest następnie przetwarzany a wynik tej operacji jest prezentowany w prawym panelu. Dodatkowo na dole wypisywany jest komunikat z odpowiedzią jaką uzyskaliśmy od sieci neuronowej. W algorytmie na podstawie testów został określony próg poniżej którego odpowiedź sieci zostaje uznana jako ’gest nierozpoznany’.



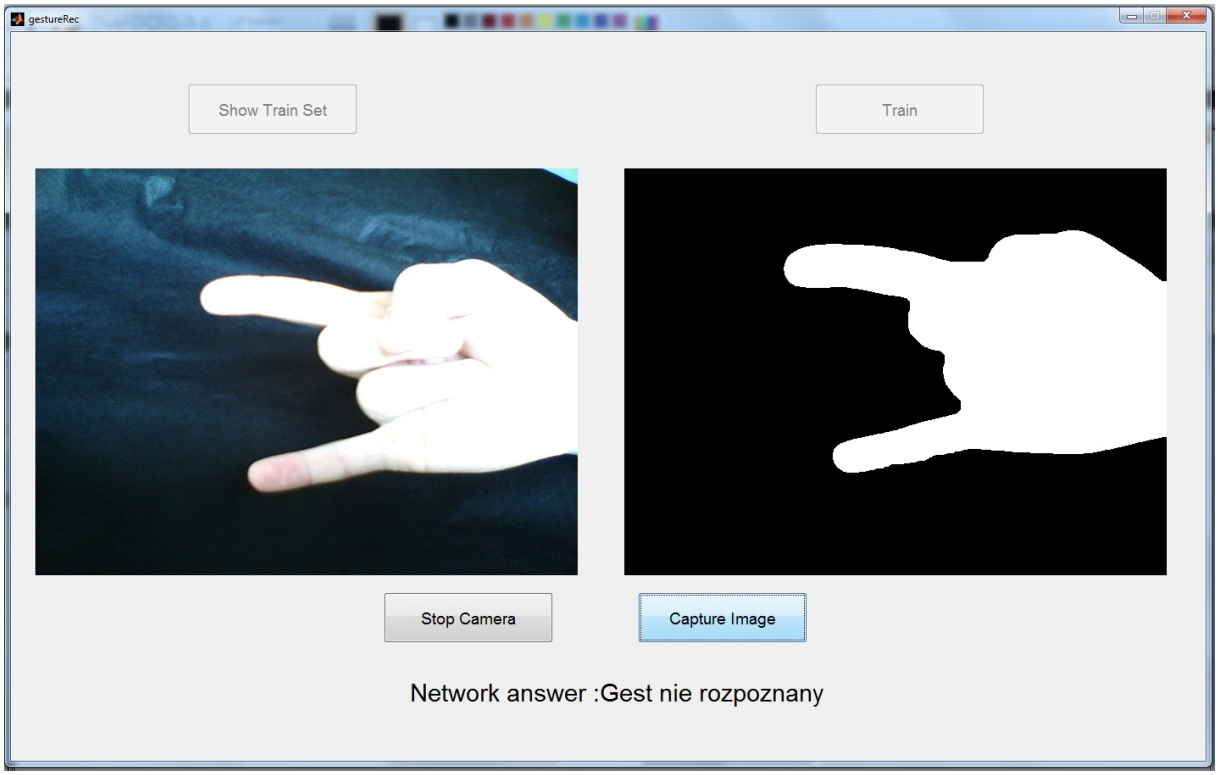
Rysunek 2.2: Rozpoznanie gestu otwartej dłoni



Rysunek 2.3: Rozpoznanie gestu pięści



Rysunek 2.4: Rozpoznanie gestu znaku ’V’



Rysunek 2.5: Nierozpoznanie gestu

3. Wnioski

W niniejszym projekcie udało się stworzyć prosty interfejs użytkownika umożliwiający robienie zdjęć w czasie rzeczywistym, które następnie zostają przetworzone przy użyciu odpowiednich transformacji i operacji morfologicznych. Tak przygotowane zdjęcia zostają wrzucone na wejście sieci neuronowej (uprzednio wytrenowanej przy użyciu zestawu uczącego), która rozpoznaje uchwycony na obrazku gest.

Aby obraz został poprawnie przetworzony użytkownik musi zapewnić jednorodne oraz ciemne tło podczas robienia zdjęć. Gdy tło jest różnorodne obraz zostaje niepoprawnie zbinaryzowany przez co gest nie może zostać rozpoznany. Jeżeli obraz był poprawnie zbinaryzowany reszta przeprowadzonych na nim operacji w większości przypadków przebiegała pomyślnie.

Sieć neuronowa w miarę dobrze rozpoznawała gesty jedynie przy dobrze przetworzonym obrazie oraz geście umieszczonym w centrum obrazu w odpowiedniej odległości od kamery . Jeżeli dłoń była umieszczona za blisko lub za daleko kamery, gest nie był poprawnie rozpoznawany.

Niestety w aplikacji nie udało się w pełni zautomatyzować algorytmu rozpoznającego typ podłączonej kamery, dlatego algorytm może nie działać przy użyciu niektórych urządzeń rejestrujących obraz. Kamery, których typ MATLAB rozpoznaje jako 'winvideo' i które obsługują rozdzielczość 640x480 powinny działać poprawnie.

Listing 3.1: Fragment kodu odpowiedzialny za połączenie z kamerą

```
1 camera = imaqhwinfo('winvideo',1);
2 supportedFormats = camera.SupportedFormats;
3 ind = cellfun(@(x)(~isempty(x)), regexp(supportedFormats,'.*640x480.*'));
4 formatName = supportedFormats(ind);
5 formatName = char(formatName);
6
7 handles.video = videoinput('winvideo', 1, formatName);
```

4. Bibliografia

1. Tadeusiewicz Ryszard, Śieci neuronowe", Warszawa : Akademicka Oficyna Wydawnicza RM, 1993
2. MATLAB NeuralNetwork Toolbox documentation
3. MATLAB ImageAquisition Toolbox documentation
4. MATLAB ImageProcessing Toolbox documentation
5. Wykłady dr. inż. Joanny Grabskiej-Chrząstowskiej
6. http://www.learnartificialneuralnetworks.com/
7. http://matlabgeeks.com/tips-tutorials/neural-networks-a-perceptron-in-matlab/